

## dlp\_hw3 实验报告 实验细节部分

钟珞妍 PB20061329

### 一、 复现结果比较

#### 1. HRSC R-50-FPN

原论文结果：

Method	Backbone	mAP(07)	mAP(12)
Oriented R-CNN	R-50-FPN	90.40	96.50

我的复现结果：

class	gts	dets	recall	ap
ship	1188	4836	0.9823	0.9034
mAP				0.9034

#### 2. HRSC R-101-FPN

原论文结果：

Method	Backbone	mAP(07)	mAP(12)
Oriented R-CNN	R-101-FPN	<b>90.50</b>	<b>97.60</b>

我的复现结果：

class	gts	dets	recall	ap
ship	1188	6234	0.9815	0.9050
mAP				0.9050

原文使用单张 RTX2080 Ti 显卡进行训练，但由于我使用的平台上 RTX2080 Ti 系列的显卡只有 11G 内存，在训练时出现了内存过载的问题，所以我的训练数据是基于 V100-32G 的。不同显卡训练的速度存在差距，但是原文并没有给出在 HRSC2016 数据集上训练的速度，所以这里只比较 mAP，从上图可以看出复现时的准确度与原文给出的基本相符。

#### 3. DOTA R-101-FPN

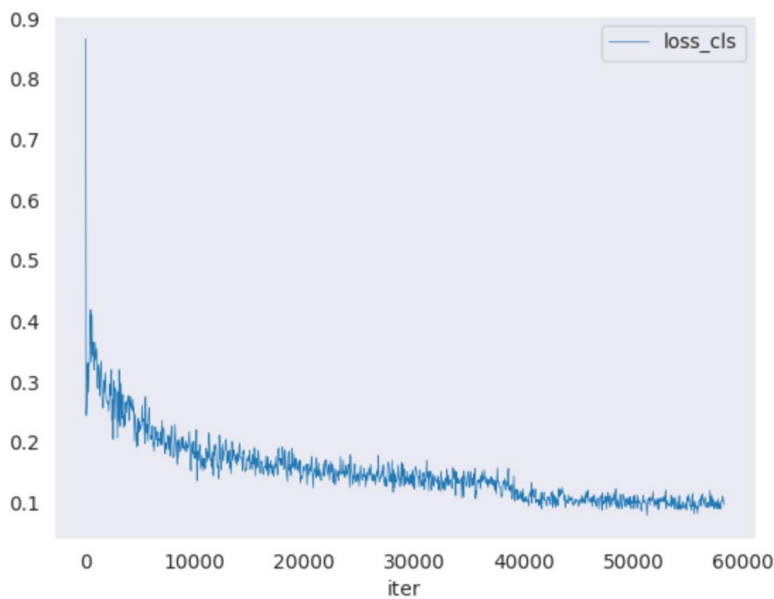
原论文结果：mAP 76.28% FPS 15.1(backbone 使用 resnet50 时)

原论文表述如下：

On the DOTA dataset, our method surpasses all comparison methods. With R-50-FPN and R-101-FPN as the backbones, our method obtains 75.87% and 76.28% mAP, presented in Table 4. All methods adopt R-50-FPN as the backbone. The hardware platform of testing is a single RTX 2080Ti with batch size of 1. During testing, the size of input images is  $1024 \times 1024$ . As shown in Table 4, our method has higher detection accuracy (75.87% mAP) than other methods but runs with comparable speed (15.1 FPS). The speed

我的复现结果：

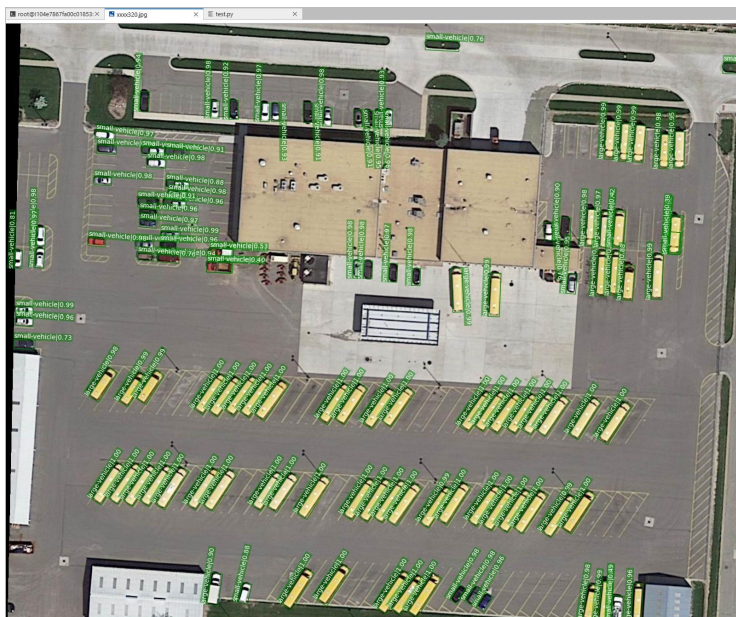
class	gts	dets	recall	ap
large-vehicle	4293	8569	0.9154	0.8437
swimming-pool	366	764	0.8251	0.6538
helicopter	72	258	0.7639	0.6006
bridge	424	2189	0.7264	0.5067
plane	2450	2844	0.9563	0.8995
ship	8861	11668	0.9593	0.8828
soccer-ball-field	87	533	0.9655	0.8001
basketball-court	124	318	0.8387	0.7266
ground-track-field	131	460	0.9008	0.7448
small-vehicle	5090	15895	0.8432	0.6734
baseball-diamond	209	463	0.8660	0.7566
tennis-court	739	943	0.9513	0.9079
roundabout	164	448	0.7866	0.6516
storage-tank	1869	3201	0.9128	0.8710
harbor	2065	3468	0.8591	0.7708
mAP				0.7527



slowest epoch 1, average time is 0.5381  
fastest epoch 12, average time is 0.5256  
time std over epochs is 0.0032  
average iter time: 0.5297 s/iter

与原文一样在单张 RTX2080 ti 上运行(可能是 batch size 或者 split 操作的原因? DOTA 数据集比 HRSC2016 大很多,但是在 11G 的 RTX2080 上跑时没有出现内存过载的情况), 便于同时比较 FPS 和 mAP, 从上图数据来看, 与原文结果基本相符。

#### 4. 实际图像展示



## 二、 对网络进行改动

### 1. 改动的位置:

将原模型 roi\_head 中用于分类分支的损失函数由 CrossEntropyLoss 改为 FocalLoss

```
roi_head=dict(
    type='OBBSStandardRoIHead',
    bbox_roi_extractor=dict(
        type='OBBSingleRoIExtractor',
        roi_layer=dict(type='RoIAlignRotated', out_size=7, sample_num=2),
        out_channels=256,
        extend_factor=(1.4, 1.2),
        featmap_strides=[4, 8, 16, 32]),
    bbox_head=dict(
        type='OBBSHared2FCBBoxHead',
        start_bbox_type='obb',
        end_bbox_type='obb',
        in_channels=256,
        fc_out_channels=1024,
        roi_feat_size=7,
        num_classes=1,
        bbox_coder=dict(
            type='OBBS2OBBDeltaXYWHTCoder',
            target_means=[0., 0., 0., 0., 0.],
            target_stds=[0.1, 0.1, 0.2, 0.2, 0.1]),
        reg_class_agnostic=True,
        loss_cls=dict(
            type='FocalLoss',
            use_sigmoid=True,
            gamma=2.0,
            alpha=0.25,
            loss_weight=1.0),
        loss_bbox=dict(type='SmoothL1Loss', beta=1.0,
            loss_weight=1.0)))
```

### 2. 改动原因:

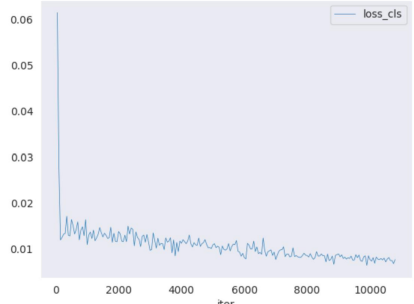
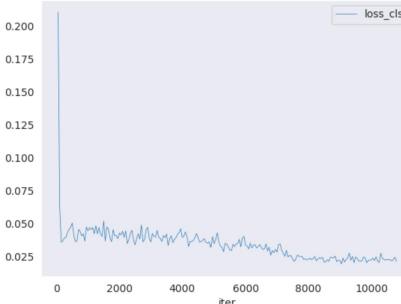
focal loss 相比于 cross entropy loss 而言，从样本难易角度出发，能在一定程度上解决样本不平衡带来的模型训练问题。

focal loss 形式为：

$$L_{fl} = \begin{cases} -\alpha(1 - \hat{y})^\gamma \log \hat{y}, & \text{当 } y = 1 \\ -(1 - \alpha)\hat{y}^\gamma \log(1 - \hat{y}), & \text{当 } y = 0 \end{cases}$$

例如若负样本远比正样本多，那么模型会倾向于数目多的负类，但是此时父类的  $\hat{y}^\gamma$  会很小，而正类的会很大，所以此时模型就会用更大的精力关注正样本。

### 3. 结果比对：

	FocalLoss	CrossEntropyLoss																														
epoch_5	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>4850</td><td>0.7239</td><td>0.6411</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.6411</td></tr></table>	class	gts	dets	recall	ap	ship	1188	4850	0.7239	0.6411	mAP				0.6411	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>4156</td><td>0.7222</td><td>0.5861</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.5861</td></tr></table>	class	gts	dets	recall	ap	ship	1188	4156	0.7222	0.5861	mAP				0.5861
class	gts	dets	recall	ap																												
ship	1188	4850	0.7239	0.6411																												
mAP				0.6411																												
class	gts	dets	recall	ap																												
ship	1188	4156	0.7222	0.5861																												
mAP				0.5861																												
epoch_10	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>6111</td><td>0.9217</td><td>0.8266</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.8266</td></tr></table>	class	gts	dets	recall	ap	ship	1188	6111	0.9217	0.8266	mAP				0.8266	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>4708</td><td>0.9108</td><td>0.8204</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.8204</td></tr></table>	class	gts	dets	recall	ap	ship	1188	4708	0.9108	0.8204	mAP				0.8204
class	gts	dets	recall	ap																												
ship	1188	6111	0.9217	0.8266																												
mAP				0.8266																												
class	gts	dets	recall	ap																												
ship	1188	4708	0.9108	0.8204																												
mAP				0.8204																												
epoch_20	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>7477</td><td>0.9453</td><td>0.8724</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.8724</td></tr></table>	class	gts	dets	recall	ap	ship	1188	7477	0.9453	0.8724	mAP				0.8724	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>3892</td><td>0.9764</td><td>0.8854</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.8854</td></tr></table>	class	gts	dets	recall	ap	ship	1188	3892	0.9764	0.8854	mAP				0.8854
class	gts	dets	recall	ap																												
ship	1188	7477	0.9453	0.8724																												
mAP				0.8724																												
class	gts	dets	recall	ap																												
ship	1188	3892	0.9764	0.8854																												
mAP				0.8854																												
epoch_36	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>5254</td><td>0.9731</td><td>0.8975</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.8975</td></tr></table>	class	gts	dets	recall	ap	ship	1188	5254	0.9731	0.8975	mAP				0.8975	<table><tr><th>class</th><th>gts</th><th>dets</th><th>recall</th><th>ap</th></tr><tr><td>ship</td><td>1188</td><td>4529</td><td>0.9756</td><td>0.9032</td></tr><tr><td>mAP</td><td></td><td></td><td></td><td>0.9032</td></tr></table>	class	gts	dets	recall	ap	ship	1188	4529	0.9756	0.9032	mAP				0.9032
class	gts	dets	recall	ap																												
ship	1188	5254	0.9731	0.8975																												
mAP				0.8975																												
class	gts	dets	recall	ap																												
ship	1188	4529	0.9756	0.9032																												
mAP				0.9032																												
loss_cls																																
time	slowest epoch 6, average time is 0.3065 fastest epoch 25, average time is 0.2934 time std over epochs is 0.0029 average iter time: 0.3002 s/iter	slowest epoch 33, average time is 0.3048 fastest epoch 16, average time is 0.2963 time std over epochs is 0.0022 average iter time: 0.3001 s/iter																														

从上方表格可以看到：focal loss 的收敛速度更快，在 epoch 较少的时候 mAP 和 recall 值会比 cross entropy loss 高出一些，在 epoch 较大的时候，二者结果基本相同，focal loss 会略差一些。



### 三、 难点与解决方案

#### 1. 选论文:

① 一开始不知道选什么样的论文, 后面找不到满足需求的论文

**难点:** 刚开始选论文的时候有点像无头苍蝇, 不知道往哪个方向选, 每天摸摸这边, 瞅瞅那边, 还是啥都不会。后面开始根据作业要求以及自己的水平去限定选择的范围, 然后发现经典且容易上手且数据集较小且代码完整的论文就找不到了。

**解决方案:** github、csdn、b 站上面能找到一些大佬们分享的“XX 领域必读论文 XX 篇”之类的链接, 后期主要是在这些里面筛选一些数据集较小, 看起来人畜无害的文章。

在作业做到后期的时候发现其实适合复现的文章有很多, 在对这个方向常用的框架有一定了解后看到的会多一些。

#### 2. 复现论文:

① 配环境:

**难点:** 我选的这篇文章要先安装两个作者写的包以及一个现有的模块, 这两个包对 python、pytorch 的版本都有一定的要求, 而 bitahub 上的镜像有限, 所以我大概零零散散花了 3-4 天的时间尝试在 bitahub 上配这个程序所需的环境, 最终无法解决这个问题, 所以自掏腰包在恒源云平台上复现的代码。

**解决过程:**

在 bitahub 上一开始我使用的是 python3.6+pytorch1.7.0

但是在安装 BboxToolkit 的时候报错提示需要 python 版本  $\geq 3.8$ :

```
(from https://pypi.org/simple/matplotlib/) (requires-python: $\geq 3.8$ )
Given no hashes to check 0 links for project 'matplotlib': discarding no candidates
ERROR: Could not find a version that satisfies the requirement matplotlib $\geq 3.4$  (from BboxToolkit==1.1) (from versions: 0.86, 0.86.1, 0.86.2, 0.91.0, 0.91.1, 1.0.1, 1.1.0, 1.1.1, 1.2.0, 1.2.1, 1.3.0, 1.3.1, 1.4.0, 1.4.1rc1, 1.4.1, 1.4.2, 1.4.3, 1.5.0, 1.5.1, 1.5.2, 1.5.3, 2.0.0b1, 2.0.0b2, 2.0.0b3, 2.0.0b4, 2.0.0rc1, 2.0.0rc2, 2.0.0, 2.0.1, 2.0.2, 2.1.0rc1, 2.1.0, 2.1.1, 2.1.2, 2.2.0rc1, 2.2.0, 2.2.2, 2.2.3, 2.2.4, 2.2.5, 3.0.0rc2, 3.0.0, 3.0.1, 3.0.2, 3.0.3, 3.1.0rc1, 3.1.0rc2, 3.1.0, 3.1.1, 3.1.2, 3.1.3, 3.2.0rc1, 3.2.0rc3, 3.2.0, 3.2.1, 3.2.2, 3.3.0rc1, 3.3.0, 3.3.1, 3.3.2, 3.3.3, 3.3.4)
```

尝试在 setup.py 文件中自行更改其中对 matplotlib 版本的要求, 但是装上之后发现无法导入这个模块。

后来使用 python3.8+pytorch1.11/pytorch1.12

但是 mmdcv 这个模块可能需要 pytorch 版本  $\leq 1.7.0$ , 由于 pytorch 版本太高, 在安装 mmdcv 时会卡住然后报错

```
Building wheels for collected packages: mmdcv-full
  Building wheel for mmdcv-full (setup.py) ... |
```

之后, 我放弃了 bitahub, 在恒源云平台上找到

python3.8+pytorch1.7.0+cu110 的镜像复现的代码

② 导入包:

**问题 1:** .py 文件中有 import 语句, \_\_init.py\_\_ 文件中有对应的文件以及函数名, 但是在运行作者的 demo 时报错找不到 BboxToolkit 和 mmdet 这两个包。

**解决过程 1:**

搜索发现一般报错 ModuleNotFound 可能是如下几个原因:

a) module 包没安装; b) 没有 import; c) 没有 \_\_init.py\_\_ 文件; d) 安装的第三方包的版本不对; e) 没设置 pythonpath 环境变量。

在排查后发现可能是 pythonpath 没设置的原因

先输入命令 `python3 -c 'import sys;print(sys.path)'` 查看在执行 `import` 语句时会搜索的真实路径，发现确实没有我安装的两个包的路径，通过 `export PYTHONPATH` 添加上。

## 问题 2：重名问题

### 解决过程 2：

之前在装 `mmcv-full` 的时候因为一直没装上，所以查看了 `mmdetection` 的说明文档，看到了有 `mmdet` 这个东西，然后在正式配环境的时候就稀里糊涂的既装了 `mmcv-full`，又装了 `mmdet`，但是在本文中作者是对 `mmdet` 进行了一些修改的，所以作者代码中有一个自己修改过的 `mmdet` 文件夹，需要通过 `python setup.py develop` 安装上，我也同时执行了这个指令，导致我的环境中有两个 `mmdet`，出现了重名问题。

## ③ 数据集：

**难点 1：**本文共使用到了两个数据集 HRSC2016 和 DOTA，HRSC2016 数据集较为简单，在处理的时候没有遇到太多问题，但是由于 DOTA 中放的是较大的航空图像，所以在使用前需要先进行 `split` 操作。

**解决过程 1：**参考官方说明文档，执行 `img_split.py` 文件，指定路径参数，可得到需要的文件。

**难点 2：**我对 DOTA 数据集存在一些疑问，DOTA 数据集下有 3 个文件夹，分别是 `train` `val` `test`，一般而言，`train` 是训练集，`val` 是训练过程中的测试集，`test` 是训练模型结束后用于评价模型结果的测试集。但是 DOTA 下的 `test` 内并没有 `labelTxt` 文件，对 `test` 内的图片进行 `split` 操作后进行测试得到的 `mAP` 和 `recall` 都为 0。

**解决过程 2：**参考 HRSC2016 中的文件结构，我发现 DOTA 下 `val` 文件夹下的内容和一般的 `test` 文件夹内的内容比较相似，所以最终测试是对 `val` 中的图片进行的操作。感觉这波操作还是挺可疑的，但是跑出来结果和原作者差不多，也拿不太准是不是该这样处理。

## 3. 改动网络：

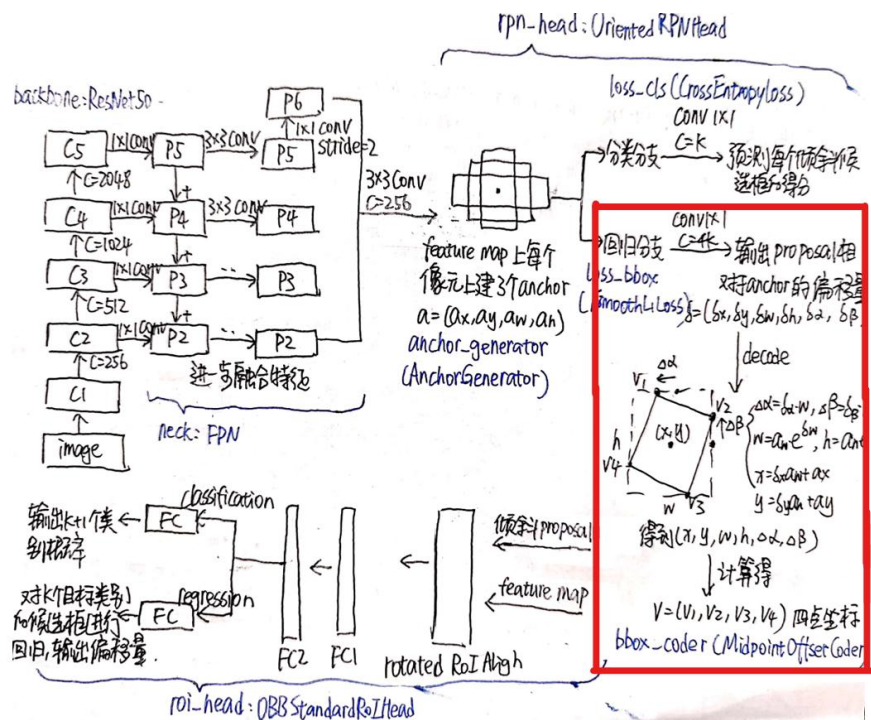
### ① 看懂作者代码结构

**难点 1：**最大的难点大概是作者代码里面的文件数量实在是太多，在最初看代码的时候根本找不到作者写的模型和方法是放在哪个文件夹下了。

**解决过程 1：**搜索的时候发现作者的代码其实是构建在 `mmdetection` 这个框架下的，文件数量多也是因为这个框架里面囊括了很多种模型。根据 `mmdetection` 的说明文档可以发现模型组件大致有 `backbone`，`neck`，`head`，`roi extractor`，`loss` 这 5 种类型，再依次去找作者的实现方法时就会容易许多。而且也由于 `mmdetection` 框架下内容很丰富，所以调用任何一个模型都比较方便。

### ② 修改网络结构

**难点：**开始时的思路是将 `rpn_head` 的部分进行一些修改，因为作者主要的贡献是提出了一种新的 `bbox_coder` 的表示方法：`MidpointOffsetCoder`，所以我打算将这一部分改为较为传统的 `DeltaXYWHTCoder`，与原文做一个对比，即对应下图中红色框内部分：



为了得到这种改变，我做了如下尝试：

- ① 仿照 S2A net 中 ARN 的结构修改 rpn\_head 中的内容，使用 XYWHABBoxCoder 表示有向候选框。

```
rpn_head=dict(
    type='RetinaHead',
    num_classes=1,
    in_channels=256,
    stacked_convs=2,
    feat_channels=256,
    anchor_generator=dict(
        type='AnchorGenerator',
        scales=[8],
        ratios=[1.0], # 原模型在每个锚点生成3种尺度的候选框，现在改为1种
        strides=[4, 8, 16, 32, 64]),
    bbox_coder=dict(
        type='XYWHABBoxCoder', # 仿照S2Anet中有向候选框的表示法 用XYWHA来预测有向候选框
        angle_range=angle_version,
        norm_factor=1,
        edge_swap=False,
        proj_xy=True,
        target_means=[.0, .0, .0, .0, .0], # 由6个参数减少为5个参数
        target_std=[1.0, 1.0, 1.0, 1.0, 1.0]),
    loss_cls=dict(
        type='FocalLoss', # 分类分支选用FocalLoss作为损失函数
        use_sigmoid=True,
        gamma=2.0,
        alpha=0.25,
        loss_weight=1.0),
    loss_bbox=dict(type='SmoothL1Loss', beta=0.11, loss_weight=1.0)),
```

- ② 观察到 retinet\_obbb 中好像使用了类似的方法，所以想把 retinet\_obbb 中 bbox\_head 的内容迁移过来：

```

bbox_head=dict(
    type='OBBRetinaHead',
    num_classes=1,
    in_channels=256,
    stacked_convs=4,
    feat_channels=256,
    anchor_generator=dict(
        type='Theta0AnchorGenerator',
        octave_base_scale=4,
        scales_per_octave=3,
        ratios=[0.5, 1.0, 2.0],
        strides=[8, 16, 32, 64, 128]),
    bbox_coder=dict(
        type='OBB2OBBDeltaXYWHTCoder',
        target_means=[.0, .0, .0, .0, .0],
        target_stds=[1.0, 1.0, 1.0, 1.0, 1.0]),
    loss_cls=dict(
        type='FocalLoss',
        use_sigmoid=True,
        gamma=2.0,
        alpha=0.25,
        loss_weight=1.0),
    loss_bbox=dict(type='L1Loss', loss_weight=1.0)))

```

由于对代码中使用到的结构了解有限，所以在进行迁移和改动的过程中出现了很多 bug，最终没有成功，但是在尝试解决问题的过程中，我学习到了有关 S2Anet 以及 Retinet 网络架构的一些知识，也对作者的代码中 rpn\_head 有关内容有了更深的了解。

**解决方法：**最终的解决方法有些逃避，即放过这个问题，选择了比较容易的更换一个损失函数，具体内容见改动网络部分。不过还是希望后面有空时再继续研究一下这个问题，看看能否解决。

#### 四、 实验总结

##### 1. 结果分析：

作业中的复现部分结果是比较好的，基本和原文中作者的表述一致。

改动部分，在最终结果上没有太显著的变化，我认为差别不显著的一个原因可能是航天图像中物体分布比较密集且数量较多，所以训练时正负样本的数量并没有很大差距。

##### 2. 心得体会：

和前 2 次作业的体会一样，难度很大，但是收获也很大。在这次作业中学到了很多东西，不仅仅是看懂了一篇论文、跑通了一份代码，更多的是从选文开始，到下载、调试程序，再到摸清代码架构，尝试修改的过程，每一步都掉了坑，但相信有了这次经历，以后可以避开很多坑。非常感谢老师和助教带来的这门课程，一定强推！